UNIVERSIDADE ESTADUAL DE CAMPINAS
FACULDADE DE TECNOLOGIA DA UNICAMP
SI201 B - Monitoria
Izael Souza

Linguagem C

Algoritmos de Ordenação

Limeira, SP - Brasil

Novembro 2022

Sumário

1 *Bubble Sort*

Bubble Sort é um dos algoritmos de ordenação mais simples que existe. A ideia dele é iterar sobre uma lista comparando os elementos adjacentes e, caso o primeiro seja maior que segundo, nos fazemos um *swap* (troca), caso queíramos ordenar a lista de forma adjacente.

Apesar de ser o mais simples, não é o mais eficiente. Como estamos comparando elementos adjacentes, nós temos que iterar a lista diversas vezes, comparando os elementos dois-a-dois. Ele é útil em casos onde a lista de elementos não é tão grande.

1.1 Explicação do Algoritmo

Nós vamos começar pelo elemento presente no *index* 0 da lista/vetor e vamos comparar com o elemento presente no *index* 1. Caso o primeiro elemento seja maior, nós vamos trocar ele com o segundo elemento. Após isso, nós vamos comparar o elemento do *index* 1 com o elemento no *index* 2. Se for maior, trocamos, caso contrário, apenas seguimos com o algoritmo. A imagem abaixo ilustra melhor essa ideia.

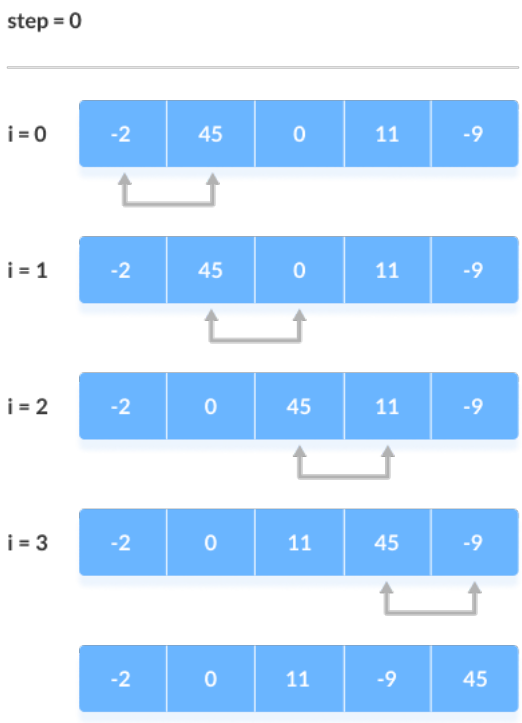

Figura 1: Comparação de elementos adjacentes no *bubble sort*. Fonte: Programiz

1.2 Implementação Iterativa

```
void bubbleSort(int* vector, int numberOfElements) {
    while(n > 2) {
        for(int i = 0; i < numberOfElements - 1; i++) {
            if(vector[i] > vector[i + 1]) {
                int aux = vector[i];
                vector[i] = vector[i + 1];
                vector[i + 1] = aux
            }
        }
        numberOfElements--;
    }
}
```

1.3 Implementação Recursiva

```
void bubbleSort(int* vector, int numberOfElements) {
    if (numberOfElements < 2)
        return;
    for(int i = 0; i < numberOfElements - 1; i++) {
        if(vector[i] > vector[i + 1]) {
            int aux = vector[i];
            vector[i] = vector[i + 1];
            vector[i + 1] = aux
        }
    }
    bubbleSort(vector, numberOfElements-1);
}
```

1.4 Pontos Importantes

- O *for loop* vai até numberOfElements - 1 pois dentro do *if* fazemos i + 1. Caso o *loop* fosse até numberOfElements, na última posição teríamos um erro pois tentaríamos acessar uma posição que não existe;
- Precisamos criar uma variável aux para não perdermos o valor de um dos elementos;

2 *Selection Sort*

2.1 Explicação do Algoritmo

Selection Sort é um algoritmo de ordenação que roda em tempo O(n^2). A ideia por trás dele é **selecionar** o maior - ou menor, dependendo de como você deseja ordenar - valor do conjunto de dados (vetor) e colocá-lo na última posição do vetor. Esse processo se repete com os n - 1 elemetos até os dois últimos.

2.2 Implementação da Função *max()*

```
int max(int* vector, int size) {
    int max = 0;
    for(int i = 0; i < size; i++) {
        if(vector[i] > max) {
            max = i; // Index of the maximum element
        }
    }

    return max;
}
```

2.3 Implementação Iterativa

Explicação da implementação Anteriormente nós definimos a função *max()* para retornarmos o **índice** do maior elemento. Agora, nós vamos iterar sobre o nosso conjunto de dados (vetor) utilizando um *while loop*. Dentro do *while*, nós vamos pegar o índice do maior elemento e vamos trocar esse elemento com o que está na última posição. Em seguida, vamos fazer *numberOfElements - 1*, pois já sabemos que o último elemento é o maior e, portanto, não faz sentido escolhermos um outro *max* e comparar com o último elemento.

```
void selecaoIterativo(int* vector, int numberOfElements){
    while(numberOfElements > 1){
        int max = max(vector, numberOfElements);
        // Begin swap
        int aux = vector[max];
```

```
        vector[max] = vector[numberOfElements - 1];
        vector[numberOfElements - 1] = aux;
        // End swap
        print(vector, numberOfElements);
        numberOfElements--;
    }
}
```

2.4 Implementação Recursiva

```
void selecaoRecursivo(int* vector, int numberOfElements){
    if(numberOfElements < 2){
        return;
    }
    int max = max(vector, numberOfElements);
    int aux = vector[max];
    vector[max] = vector[numberOfElements - 1];
    vector[numberOfElement - 1] = aux;
    print(vector, numberOfElements);
    selecaoRecursivo(vector, numberOfElements - 1);
}
```

3 *Insertion Sort*

3.1 Explicação do Algoritmo

3.2 Implementação Iterativa

```
void insercaoIterativo(int* vector, int numberOfElements){
    int lixo = -1;
    int n = 2;
    while(n <= numberOfElements){
        int aux = vector[n - 1];
        vector[n - 1] = lixo;
        int i = n - 2;
        print(vector, n);
        while(i >= 0 && vector[i] > aux) {
            vector[i + 1] = vector[i];
            vector[i] = lixo;
            i--;
            print(vector, n);
        }
        vector[i + 1] = aux;
        print(vector, n);
        n++;
    }
}
```

3.3 Implementação Recursiva

```
void insercaoRecursivo(int* vector, int numberOfElements){
    int lixo = -1;
    if(numberOfElements < 2)
        return;

    insercaoRecursivo(vector, numberOfElements - 1)
```

```
    int aux = vector[n - 1];
    int n = 2;
    int i = n - 2;
    print(vector, n);

    while(i >= 0 && vector[i] > aux) {
        vector[i + 1] = vector[i];
        vector[i] = lixo;
        i--;
        print(vector, n);
    }
    vector[i + 1] = aux;
    print(vector, n);
}
```